Malai : un Modèle Conceptuel d'Interaction pour les Systèmes Interactifs

Arnaud Blouin, Olivier Beaudoux

Malai : un Modèle Conceptuel d'Interaction pour les Systèmes Interactifs

Arnaud Blouin

GRI - Groupe ESEO
Angers, France
arnaud.blouin@eseo.fr

Olivier Beaudoux

GRI - Groupe ESEO
Angers, France
olivier.beaudoux@eseo.fr

RESUME

Malgré l'évolution des modèles et des techniques d'interaction, les plate-formes d'implémentation de systèmes interactifs (SI) se focalisent sur la définition d'interfaces utilisateur (IU), ignorant la notion d'action et d'instrument. Cet article présente Malai, un modèle conceptuel d'interaction dédié au développement de systèmes interactifs. Malai vise à réunir différents principes du modèle d'action de Norman, de l'interaction instrumentale, de la manipulation directe, des interacteurs et du modèle DPI. Il complète nos travaux déjà réalisés sur les techniques de manipulation de données utilisées pour lier des données sources à des IU. A l'aide d'une étude de cas, nous présentons les quatre modèles composant Malai, à savoir : les modèles d'interface, d'action, d'interaction et d'instrument. A partir de ces modèles, le code du SI peut être généré dans une approche IDM (*Ingénierie Des Modèles*).

MOTS CLES : Interaction, action, instrument, IDM

ABSTRACT

Despite the evolution of interaction, implementation frameworks for the development of Interactive Systems (IS) still focus on user interfaces (UI) where the notion of action and instrument are still ignored. This paper introduces Malai, a conceptual interaction model dedicated to the development of interactive systems. Malai aims at gathering principles of the Norman's action model, the instrumental interaction, the direct manipulation, the interactor concept and the DPI model. It completes works on data manipulation techniques used to link source data to UI. With the help of a use case, we present the four models that compose Malai : the interface model, the action model, the interaction model and the instrument model. Using these models, the SI code can be generated within an MDE approach (*Model-Driven Enginnering*).

CATEGORIES AND SUBJECT DESCRIPTORS: D.2.10 Design : Methodologies ; H.5.2 Information Interfaces and Presentation :Theory and methods

GENERAL TERMS: Design

KEYWORDS: Interaction, action, instrument, MDE

INTRODUCTION

La visualisation, la navigation et l'édition de données sont les tâches principales des Systèmes Interactifs (SI). Les plate-formes de développement modernes, tel que WPF [23], formalisent la définition du lien entre les données sources et les interfaces utilisateurs (IU) par le biais du « data binding ». Malgré l'évolution des techniques d'interaction, ces plate-formes restent toujours centrées sur le concept de widget ignorant ainsi les notions d'interaction, d'action et d'instrument. A un niveau plus abstrait, les environnements de développement fondés sur les modèles, tel que UsiXML [14], suivent, pour une majorité d'entre eux, un processus de développement composé de différents niveaux d'abstraction. Cependant, ils n'appliquent pas le principe de l'instrument, tandis que la description des interactions se limitent toujours à la notion d'évènement associé à un objet.

Le but de notre travail est de pouvoir générer une partie ou l'ensemble du code des SI à partir de modèles exhaustifs. Pour cela, nous proposons un modèle conceptuel pour la conception de SI manipulant des données. Celui-ci se compose de cinq modèles regroupés en deux parties. La première partie, appelée *Malan*, concerne le modèle des données sources que le SI manipule et son lien vers l'IU du SI [5]. La seconde partie, présentée dans cet article, consiste en un modèle conceptuel d'interaction nommé *Malai*, composé d'un modèle d'interface, d'un modèle d'action, d'un modèle d'interaction et d'un modèle d'instrument. *Malai* réunit des principes du modèle d'action de Norman [18], de l'interaction instrumentale [2], de la manipulation directe [24], du concept d'interacteur [16] et du modèle DPI [3]. *Malai* vise à retenir les avantages de ces modèles tout en apportant les améliorations suivantes :

– définir et fournir une librairie d'interactions prédéfinies pouvant être utilisées tel quel dans différents SI ;

– clarifier la définition du feedback intérimaire [16] des actions et des instruments ;
– réutiliser une même action pour différentes IU qui utilisent le même modèle de données sources ;
– clarifier l'implémentation du processus d'undo/redo des actions.

Le tout dans une approche IDM fondée sur trois des quatre composants du modèle Cameleon : l'interface abstraite (AUI) ne contient aucune information graphique du SI ; l'interface concrète (CUI) spécifie la disposition graphique des éléments de l'interface ; l'interface finale (FUI) correspond au code du SI pour une plate-forme d'application donnée [6]. L'autre composant, les concepts et les tâches qui y sont liées, n'est pour l'instant pas abordé dans Malai. Après une présentation des travaux connexes, nous décrivons les modèles d'interface, d'action, d'interaction et d'instrument qui composent Malai. Un même exemple est utilisé tout au long de l'article pour illustrer chacun des modèles.

TRAVAUX CONNEXES

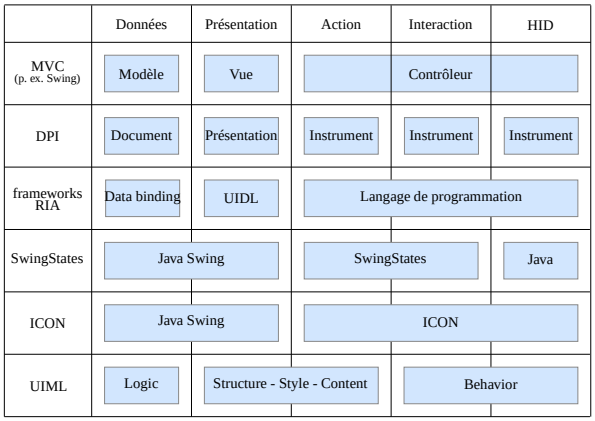

FIGURE *1* **:** Classification des travaux connexes.

La plupart des frameworks actuels sont fondés sur le modèle MVC [13]. MVC (*Modèle-Vue-Contrôleur*) est un *design pattern* séparant les données sources manipulées par le système (le modèle), des vues présentant ces données à l'utilisateur, et des contrôleurs qui gèrent les évènements en provenance des périphériques d'entrée (HID - *Human Input Devices*). Cependant, MVC ne fournit ni modèle d'interaction ni modèle d'action, comme le résume la figure 1.

Le système Garnet a introduit la notion d'interacteur pour faciliter le développement d'IU, notamment en séparant les interactions des widgets [16]. Ce principe de séparation est utilisé dans des boîtes à outils afin de rendre des interacteurs indépendants des interactions et de leur contexte d'utilisation [9]. Proche de l'interacteur, l'interaction instrumentale définit un instrument comme étant la métaphore d'un outil qu'un utilisateur manipule pour réaliser des actions [2]. Un instrument se compose d'une partie physique définissant les interactions et les HID. Sa partie logique correspond à la représentation de l'instrument dans le SI. Fondé sur l'interaction instrumentale, DPI est un modèle conceptuel orienté documents utilisant la notion d'action [18] dans un instrument [3] : la partie logique d'un instrument génère des actions consommées par l'IU.

Le formalisme ICO décrit le fonctionnement de SI à l'aide de réseaux de Petri [17]. Le fonctionnement de notre modèle d'instrument est assez proche du composant ObCS de ICO, permettant de décrire le comportement d'un objet interactif, dans le sens où des évènements HID sont utilisés en entrée et des services sont appelés en sortie. ICO ne considère cependant pas les actions et les interactions comme des objets de première classe en utilisant la notion d'évènement au lieu de la notion d'interaction et en appelant directement des méthodes du modèle de données pour modifier ces dernières.

Les récentes plate-formes de développement RIA (*Rich Internet Applications*), tel que WPF [23], sont dédiées à la création d'applications classiques (de bureau) et pour le Web, à l'aide d'un langage de description d'IU (UIDL - *User Interface Definition Language*) et d'un langage de programmation. Cependant, ces plate-formes sont centrées sur la définition de l'IU et ne fournissent, par conséquent, pas de modèle d'action, d'interaction et d'instrument. Les environnements de développement fondés sur les modèles (MB-UIDE), qui suivent généralement les quatre niveaux de conception proposés par Cameleon, visent à faciliter l'adaptation d'un SI devant s'exécuter sur différentes plate-formes d'exécution. Ces MB-UIDE, tels qu'UsiXML [27] et UIML [19], ne proposent cependant pas de modèle d'instrument et utilisent toujours la notion d'évènement pour décrire les interactions. Bien qu'UsiXML dispose d'un modèle de tâche similaire sur certains points à notre modèle d'action, ce MB-UIDE ne considère pas les actions comme des objets possédant leur propre cycle de vie géré par les instruments et en relation avec des interactions. La notion de feedback pour les tâches n'est également pas abordée.

Au contraire, d'autres boîtes à outils issues de travaux de recherche se concentrent principalement sur l'interaction. ICON (*Input CONfiguration*) est notamment une boîte à outils permettant la configuration d'interactions physiques (souris, clavier, *etc.*) et leur connexion à une IU [10]. SwingStates est une librairie ajoutant à la boîte à outils Java Swing, la description d'interactions sous la forme de machines à états, remplaçant ainsi les écouteurs habituellement utilisés [1].

VUE D'ENSEMBLE DU MODELE

Notre modèle conceptuel, dont l'organisation est illustrée par la figure 3, se compose de deux parties complémentaires. La première, appelée *Malan*, permet la définition du lien entre les *données sources* et les différentes *présentations* d'un SI, où une présentation affiche des données aux utili-

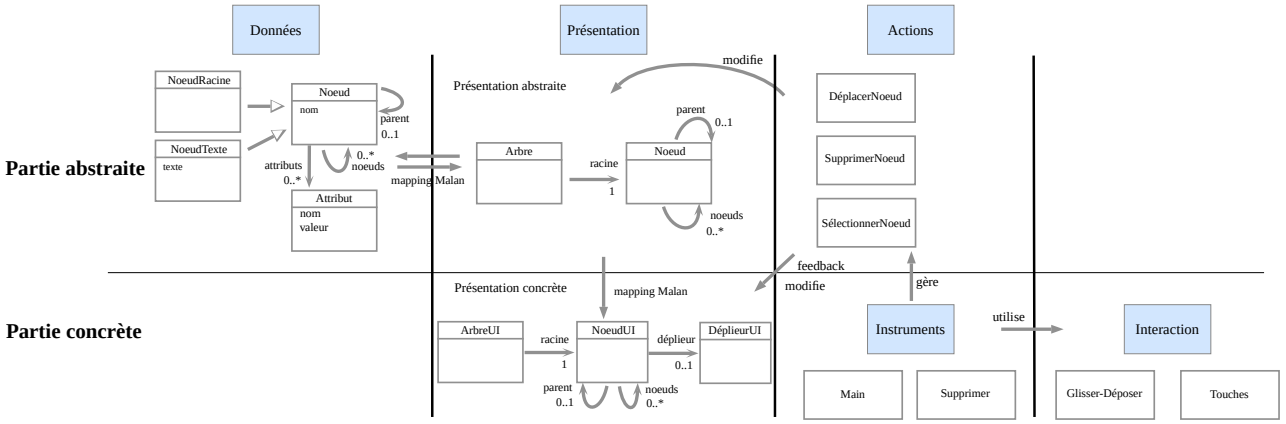

FIGURE ***2*** **:** Organisation de l'exemple de l'arbre.

sateurs [5]. La seconde partie concerne le modèle conceptuel d'interaction *Malai*, présenté dans cet article. Malai permet la définition de la facette interactive d'un SI en se basant sur les notions d'*actions* et d'*instruments*. Un instrument sert de médiateur entre l'utilisateur et le système : l'utilisateur manipule des instruments, en effectuant des *interactions* à l'aide d'*HID* afin de réaliser des actions qui modifient une présentation ou les données sources du SI. Lorsque les données sources sont modifiées, le lien entre les données et chaque présentation du SI, défini avec Malan, permet la mise à jour partielle des présentations en actualisant uniquement leurs éléments concernés par les modifications. Nous considérons une interface utilisateur comme l'ensemble des présentations et des instruments d'un SI.

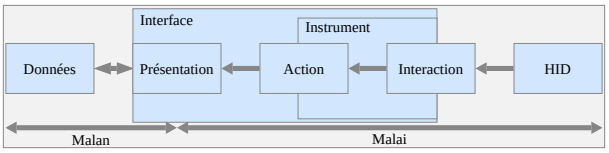

FIGURE ***3*** **:** Organisation des cinq modèles.

Les sections suivantes décrivent les éléments principaux qui composent Malai, à savoir les modèles d'interface, d'interaction, d'action et d'instrument.

ETUDE DE CAS

Tout au long de cet article, un même exemple est utilisé pour illustrer chacun des modèles de Malai. Cet exemple est une application présentant un fichier XML par le biais d'un arbre. L'utilisateur interagit avec l'application pour éditer et consulter le fichier source XML.

La figure 2 est une figure synoptique qui introduit les cinq objets centraux de l'application (les données, les présentations abstraite et concrète, les actions, les instruments et les interactions) et leurs liens. Les sections suivantes détaillent les différents modèles qui composent Malai (les modèles d'interface, d'interaction, d'action et d'instrument) en faisant référence à la figure 2 pour illustrer la partie de l'application concernée par le modèle, et en présentant du pseudo-code de l'application.

MODELE D'INTERFACE

Définition

Dans notre modèle, les notions de présentation et d'interface utilisateur sont définies de manière distincte, comme l'illustre la figure 4. Une présentation permet à des utilisateurs de visualiser des données sources de différents points de vue. Elle est composée d'éléments graphiques organisés en graphe de scène représentant l'ensemble ou une partie des données. Une interface utilisateur est composée d'une ou plusieurs présentations et d'un ensemble d'instruments. Les instruments (p. ex. les barres de défilements et les boutons) visent à faciliter la visualisation, la navigation et l'édition des données sources.

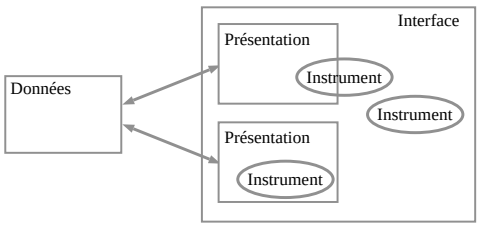

FIGURE ***4*** **:** Différences entre une interface utilisateur, une présentation et un instrument.

Comme détaillé dans les figures 3 et 4, le modèle d'interface est directement lié aux modèles d'action, d'interaction et d'instrument : la partie logique d'un instrument correspond à sa représentation dans l'interface, tandis que la partie physique concerne l'interaction que réalise l'utilisateur pour manipuler l'instrument. Un instrument déduit les actions à appliquer sur l'interface ou sur les données sources *via* les paramètres de la partie physique, comme l'expliquent les sections suivantes.

Exemple

La figure 5 présente un exemple d'interface finale de l'éditeur développée en Java. On y remarque la présence de « déplieurs » au coté de chaque nœud dépliable sous la forme de triangles. Ces widgets correspondent à la face visible de l'instrument gérant le pliage et le dépliage des nœuds. Ils illustrent bien le principe qu'un instrument peut faire partie intégrante d'une présentation concrète.

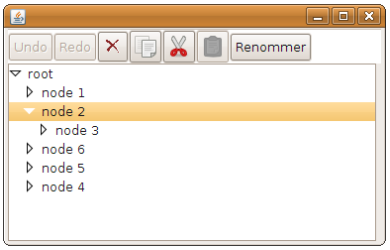

FIGURE *5* **:** L'UI de l'exemple de l'arbre.

MODELE D'INTERACTION

Définition

Une interaction est décrite par une *machine déterministe à nombre fini d'états* prenant en entrée des évènements HID pour en déduire l'état de sortie. Une interaction peut être implémentée sur une plate-forme d'exécution telle que SwingStates [1]. La modélisation à l'aide de machines à états fournit un moyen simple et efficace de décrire des comportements, en particulier des interactions homme-machine [1], et permet la spécification d'interactions complexes indépendamment des actions et des instruments des SI. Un développeur dispose ainsi d'une librairie d'interactions prédéfinies pouvant être facilement utilisées dans différents SI (p. ex. une interaction *glisser-déposer*). Il peut également définir de nouvelles interactions agrandissant ainsi cette librairie.

Cycle de vie

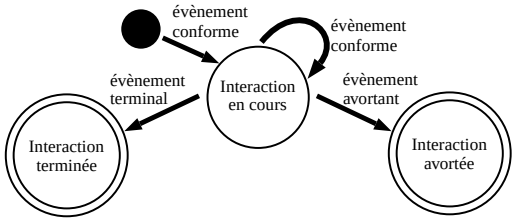

FIGURE *6* **:** Le cycle de vie d'une interaction.

La figure 6 présente le cycle de vie d'une interaction. Une interaction démarre lorsqu'un premier évènement conforme survient. Elle continue tant qu'un état final n'est pas atteint et tant qu'aucun évènement avortant n'apparait. Un évènement est dit avortant lorsqu'il stoppe l'interaction en cours (p. ex. l'appui sur la touche *'échap'* lors d'un *glisser-déposer*). Ainsi, ce processus suit la recommandation de la manipulation directe stipulant qu'un utilisateur doit pouvoir arrêter toute interaction effectuée sur un système [24].

Exemple

La figure 7 représente une machine à états décrivant une interaction *glisser-déposer* permettant l'utilisation du clavier. Cette interaction est utilisée par des instruments de l'exemple de l'arbre pour effectuer des actions, comme décrit dans les sections suivantes. Elle peut démarrer avec un évènement *pression* du dispositif de pointage, s'exécute tant que des évènements *glissement* surviennent, et se termine avec un évènement *relâchement*. Elle peut être avortée si l'utilisateur appuie sur la touche '*échap*', ou si un évènement *relâchement* apparaît alors que l'état courant est *pressé*. Les évènements *touchePressée* et *relâchementTouche*, des états *pressé* et *glissé*, permettent l'utilisation du clavier pendant l'interaction.

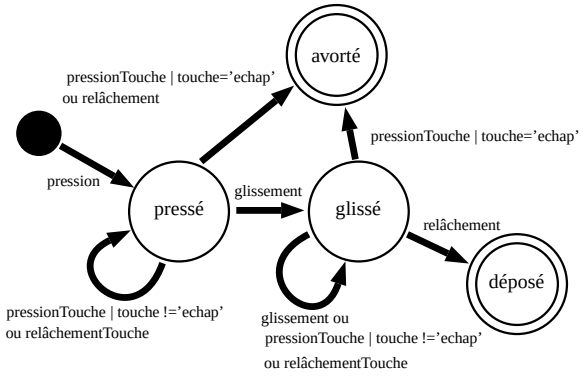

FIGURE **7:** Exemple d'une interaction *glisser-déposer* représentée par une machine à états.

Le pseudo-code suivant est un extrait de la définition de l'interaction *glisser-déposer* présentant ses attributs (lignes 2 à 6) modifiés au cours de l'interaction lors des transitions d'états (lignes 9 à 20).

```
1 Interaction glisser-déposer {
2   Point   pointInitial
3   Point   pointFinal
4   Objet   objetCible
5   Objet   objet
6   Touche  touche
7   // ...
8
9   Etat pressé{
10    Transition glisseHID -> glissé{
11      pointFinal = getPoint()
12      objetCible = getObjet(pointFinal)
13      notifier que l'interaction évolue
14    }
15
16    Transition pressionTouche='échap' -> pressé{
17      touche = getTouche()
18      notifier que l'interaction évolue
19    }
20    // ...
21  }
22 }
```

MODELE D'ACTION

Définition

Une action résulte du besoin pour un utilisateur ou un système de réaliser une tâche ou une partie d'une tâche soit sur des données sources *via* une IU, soit sur une IU sans modifier les données. Il existe deux types d'actions : les actions utilisateurs et les actions systèmes. Les actions utilisateurs sont produites par des instruments que manipule un utilisateur par le biais d'interactions. Au contraire, les actions systèmes sont lancées par le système lui-même et ne dépendent d'aucun instrument ou interaction (p. ex. la recherche de mises à jour d'un logiciel). La spécification d'une action s'effectue indépendamment des instruments qui peuvent la lancer.

Notre modèle d'action diffère des modèles de tâche, tel que celui d'UsiXML, principalement du fait d'une différence de niveau d'abstraction : un modèle de tâche est généralement utilisé pour générer une interface abstraite, or notre modèle d'action opère au niveau de celle-ci. Un modèle d'action Malai enrichit un modèle de tâche en y apportant le feedback intérimaire et le code des méthodes `faire`, `peutFaire`, *etc.* des actions. De plus, une action est considérée comme un objet possédant son propre cycle de vie géré par l'instrument, comme le détaille les sections suivantes. Dans nos travaux futurs, nous envisageons de générer un modèle d'action à partir d'un modèle de tâche.

Spécification d'actions

Une action est définie par une classe UML dans laquelle les attributs décrivent l'état de l'action. En utilisant l'exemple de l'arbre, l'action `SélectionnerNoeud` définit l'attribut `sélection` correspondant au nœud sélectionné de la présentation, comme l'illustre la figure 8.

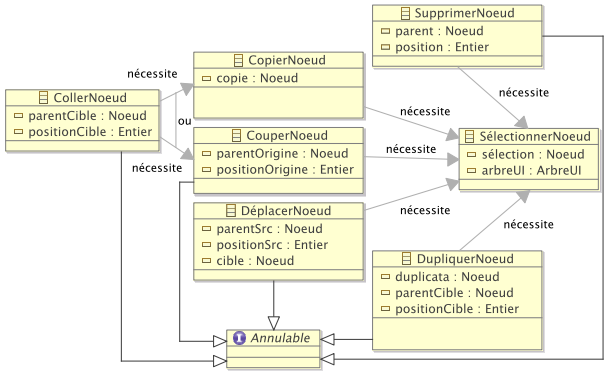

FIGURE ***8*** **:** Exemple d'actions de l'exemple de l'arbre.

La création d'une action peut nécessiter l'existence d'autres actions déjà exécutées. Par exemple (figure 8), un nœud de l'arbre doit être d'abord sélectionné avant d'être déplacé, copié, coupé ou dupliqué. Une action peut également nécessiter telle ou telle action : pour coller un nœud dans l'arbre, il est nécessaire d'avoir, au préalable, un nœud copié *ou* collé.

Cycle de vie

La figure 9 correspond au cycle de vie d'une action. Il étend le cycle de vie une action du design pattern Commande dans lequel les actions sont des objets parametrables et dont les effets peuvent être annulée [11]. Bien que la définition d'une action soit indépendante des interactions et des instruments, le cycle de vie d'une action se déroule en relation avec celui de l'interaction, grâce à l'instrument qui leur sert de connecteur.

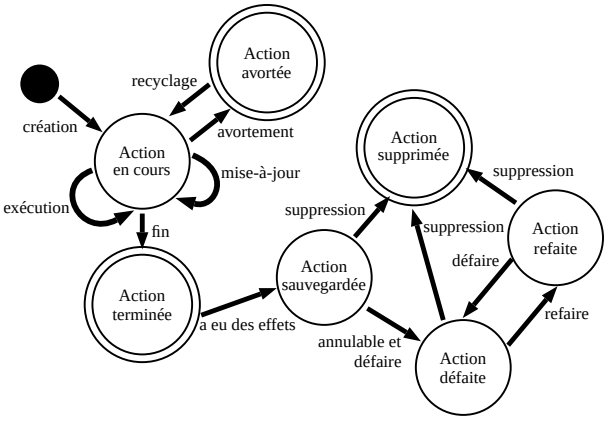

FIGURE ***9*** **:** Le cycle de vie d'une action.

Une fois créée, une *action en cours* peut être *exécutée* et *mise à jour* plusieurs fois. Une action en cours peut également être *avortée* pour être éventuellement *recyclée* en une autre action qui devient la nouvelle action en cours. Une fois l'état terminal *Action terminée* atteint, si l'exécution de l'action a eu des effets sur les données ou sur l'IU, elle est mémorisée pour être éventuellement défaite et refaite. Sinon, le cycle de vie de l'action se termine. Une action mémorisée peut être supprimée du système (l'état terminal *Action supprimée*). Ce cas de figure peut par exemple survenir lorsque que la taille de la mémoire dédiée aux opérations undo/redo est limitée.

Exécution et undo/redo L'exécution d'une action peut modifier :

– une présentation sans modifier les données sources. La sélection d'un nœud d'un arbre, par exemple, ne modifie pas le document source XML, mais influe sur sa présentation en mettant en valeur le nœud concerné. Le pseudo-code suivant définit l'action `SélectionnerNoeud` qui possède deux attributs `arbreUI` et `sélection` correspondant respectivement à la présentation sous forme d'arbre des données sources et au nœud sélectionné de la présentation. La méthode `faire` (ligne 4 à 9) permet l'exécution de l'action : si l'action peut être exécutée, l'instance de `NoeudUI` correspondant à l'instance `Noeud` choisi est sélectionnée.

```
1 Action SelectionnerNoeud {
2   Noeud    sélection
3   ArbreUI arbreUI
4   faire() {
5      arbreUI.selectionnerNoeud(
6        #obtenirCible(NoeudUI, sélection))
7   }
8   booléen peutFaire() {
9    retourner sélection!=null et
10             arbreUI!=null}
11 }
```

– les données sources. Dans ce cas, les présentations doivent être partiellement mises à jour en actualisant uniquement leurs éléments concernés par les modifications. Par exemple, l'action `SupprimerNoeud` supprime un nœud du document source XML. Par conséquent, la présentation concrète (`ArbreUI`) doit supprimer son nœud correspondant au nœud du document XML, comme l'illustre le pseudo-code suivant, définissant l'action `SupprimerNoeud`. L'exécution de cette action (lignes 6 à 12) sauvegarde le nœud à supprimer, puis l'efface des données sources. Cette action est à même d'être défaite (ligne 14) et refaite (ligne 18), et nécessite une action `SélectionnerNoeud` pour exister (la variable `sélection` provient de l'action `SélectionnerNoeud`).

```
1 Action SupprimerNoeud necessite
2      SelectionnerNoeud est Annulable{
3   Noeud  parent
4   Entier position
5
6   faire() {
7     si(peutFaire()) {
8       parent      = sélection.parent
9       position    = parent.position(noeud)
10      #supprimer(parent.noeuds, sélection)
11    }
12  }
13  peutFaire() { retourner sélection!=null }
14  defaire() {
15    #ajouter(parent.noeuds, sélection,
16                            position)
17  }
18  refaire() {
19    #supprimer(parent.noeuds, sélection)
20  }
21 }
```

Une fois exécutée et si elle est annulable, une action peut être défaite et refaite par un processus "undo/redo" standard.

Avortement et recyclage Il est possible d'avorter une action en cours. Dans ce cas, elle peut être recyclée en une action différente si certaines conditions sont respectées. Cela évite ainsi à l'utilisateur de recommencer le processus de création de cette dernière action. Cela permet également de pouvoir changer d'action au cours de l'interaction. Le recyclage est géré par l'instrument qui sert de lien entre l'interaction (et donc l'utilisateur) et l'action (l'interface et les données sources). Pour informer l'utilisateur de l'action en cours, et par conséquent du passage d'une action à une autre, l'instrument doit fournir un feedback intérimaire.

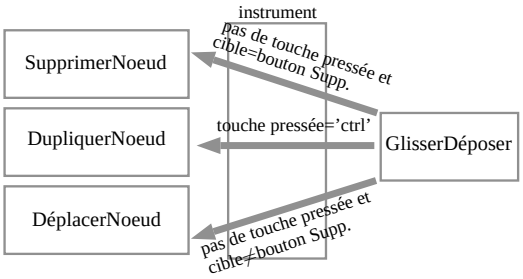

FIGURE ***10*** **:** Exemple de recyclage d'actions.

La figure 10 décrit les liaisons pondérées par des conditions, gérées par un instrument, entre l'interaction « glisser-déposer » de la figure 7 et les actions `DéplacerNoeud`, `SupprimerNoeud` et `DupliquerNoeud`. Lorsqu'un utilisateur fait glisser un nœud d'un arbre, une action `DéplacerNoeud` est créée, étant donné que la condition « pas de touche et cible≠bouton Supp. » est respectée. Si l'utilisateur appuie sur la touche « ctrl », l'action en cours `DéplacerNoeud` est alors avortée et recyclée en une action `DupliquerNoeud` puisque les paramètres de l'interaction correspondent maintenant à la condition « touche pressée='ctrl' ». Enfin, si la cible du « glisser-déposer » est le bouton `Supprimer` de l'IU (voir la figure 5), l'action correspondante est maintenant du type `SupprimerNoeud`, recyclée à partir de l'action `DupliquerNoeud`. Ce processus continue jusqu'à ce que l'interaction se termine ou soit avortée.

Feedback intérimaire de l'action

Le feedback intérimaire consiste à fournir à l'utilisateur une réponse continue aux actions qu'il effectue sur le système [25]. Cela se résume à présenter à l'utilisateur le résultat que provoquerait l'exécution de l'action courante [16]. Le feedback intérimaire d'une action peut être considéré indépendamment des instruments puisque que plusieurs instruments sont capables de créer une même action et que le feedback intérimaire de l'action reste toujours le même.

Le pseudo-code suivant définit le feedback intérimaire de l'action `DéplacerNoeud` (*cf.* figure 8). Il consiste à montrer à l'utilisateur le résultat du déplacement en cours en ajoutant un nœud dans l'arbre (la présentation) et à le rendre transparent.

```
1 InterimFeedback de DeplacerNoeud {
2   DeplacerNoeud action
3   ArbreUI     arbre
4   NoeudUI     noeudTemp
5
6   interimFeedback() {
7     si(action.peutFaire() et !action.fait()) {
8       Entier    pos
9       NoeudUI   cible
10      si(noeudTemp==null) {
11        noeudTemp = nouveau NoeudUI(
```

```
12                          action.sélection)
13          noeudTemp.définirOpacité(0.5)
14        }
15        si(action.cible.peutContenir()) {
16          cible = #obtenirCible(NoeudUI,
17                          action.cible)
18          pos    = 0
19        }sinon {
20          cible = #obtenirCible(NoeudUI,
21                          action.cible.parent)
22          pos    = cible.position(action.cible)
23        }
24        noeudTemp.deplacer(cible, position)
25      }
26      sinon noeudTemp.supprimer()
27    }
28 }
```

MODELE D'INSTRUMENT

Définition

L'instrument est un « médiateur entre l'utilisateur et les données. L'utilisateur agit sur un instrument qui transforme l'interaction réalisée par l'utilisateur en actions s'appliquant sur les données sources ou sur l'IU »[2] (*cf.* figure 11). Les instruments fournissent un feedback intérimaire permettant à l'utilisateur de disposer des informations relatives à l'état de l'instrument et des actions que ce dernier peut permettre de créer.

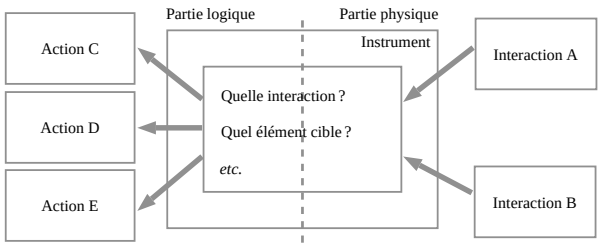

FIGURE *11* **:** Le passage d'une interaction à une action dans un instrument.

Un instrument se compose d'une partie physique décrivant les interactions utilisées en entrée, et d'une partie logique définissant la présentation de l'instrument dans l'IU. Un instrument est capable de créer plusieurs actions suivant le contexte courant (type de l'interaction, ses attributs, *etc.*). En fonction de l'état d'une interaction en cours qu'un instrument écoute, ce dernier crée, met à jour, exécute, termine, avorte ou recycle l'action correspondante.

Processus global

Un instrument est lié à un ensemble d'interactions dont il attend des changements d'états. Ces changements sont le démarrage, la mise à jour, la fin et l'avortement d'une interaction. Pour chacun de ces types de changement, un instrument peut :

– Fournir du feedback intérimaire. Le feedback intérimaire d'un instrument donne un aperçu à l'utilisateur de l'évolution de l'interaction et de l'instrument. Par exemple, la figure 12 présente une partie d'une IU d'un logiciel de dessin développé avec notre prototype. Cette IU montre que l'utilisateur réalise un *glisser-déposer* de la couleur du bouton `Remplir`, vers la forme ciblée. Dans cet exemple, le feedback intérimaire d'un instrument consiste à associer la couleur du bouton au curseur pendant le *glisser-déposer*. Ce processus permet à l'utilisateur d'être informé de l'état courant de l'instrument qu'il manipule et de l'action courante que de ce dernier peut créer.

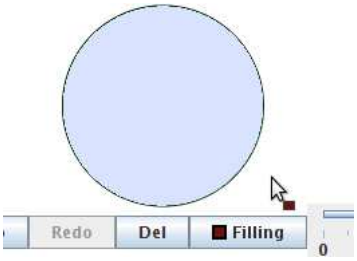

FIGURE *12* **:** Un exemple de feedback intérimaire d'un instrument.

– Gérer (créer, avorter, mettre à jour, exécuter, recycler et terminer) l'action correspondante. L'état de l'action est déduit à partir d'un ensemble de paramètres, tels que le type ou les paramètres de l'interaction en cours.

Un instrument peut être activé et désactivé. Par exemple, lorsqu'un nœud est sélectionné, l'instrument `Suppression`, qui gère la suppression des nœuds, est activé et *vise versa*. Un développeur dispose de deux manières de gérer l'activation d'un instrument. Tout d'abord, si aucune action que nécessite les actions que créé un instrument n'existe, alors l'instrument peut être désactivé. Par exemple, l'instrument Suppression ne peut produire que l'action `SupprimerNoeud` qui nécessite l'action `SélectionnerNoeud` ; durant l'exécution du SI, si aucun nœud n'est sélectionné, alors l'action `SupprimerNoeud` ne peut exister et l'instrument `Supprimer` est désactivé. Ensuite, des actions génériques `ActiverInstrument` et `DesactiverInstrument` peuvent être créées par des instruments pour en (dés-)activer d'autres.

Nous considérons les boîtes de dialogue comme étant des instruments pouvant contenir d'autres instruments. Par exemple, une palette de couleur est la présentation concrète d'un instrument visant à choisir une couleur. L'affichage de la palette s'effectue en (dés-)activant l'instrument.

Exemple

La figure 13 présente un instrument décrivant en partie l'instrument *Main* de notre exemple de l'arbre. Le but de cet instrument est de manipuler les nœuds de l'arbre en appliquant le principe de la manipulation directe.

Cet instrument utilise deux interactions différentes : l'interaction *glisser-déposer* de la figure 7 et l'interaction

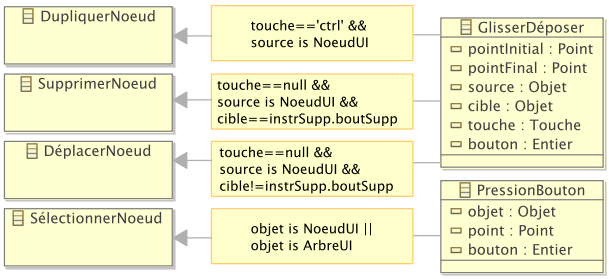

FIGURE ***13*** **:** Un exemple d'instrument.

pressionHID correspondant, par exemple, à la pression d'un bouton d'une souris. Pour chacune de ces interactions, l'instrument calcule l'action correspondante à gérer en fonction de conditions (p. ex. *objet is NoeudUI* || *objet is ArbreUI*). Le pseudo-code suivant décrit l'instrument *Main* composé d'un attribut `action` et d'un autre `interaction` correspondant respectivement à l'action et l'interaction en cours lorsque l'instrument est actif. La fonction `recyclageActions` (ligne 5) correspondant au processus de recyclage décrit précédemment.

```
1 Instrument Main {
2  Action action
3  Interaction inter
4
5 recyclageActions(Interaction i) {
6   Booleen recycle = faux
7   si(i est GlisserDeposer) {
8     Objet o = objet cible de i
9     si((action est DeplacerNoeud ou
10        action est DupliquerNoeud) et
11        o est bouton "Supp.") {
12       avorter l'action en cours
13       créer une action SupprimerNoeud
14       recycle=vrai
15    }
16    sinon si(action est SupprimerNoeud et
17             o n'est pas bouton "Supp.") {
18       avorter l'action en cours
19       recycle=vrai
20       si(touche de i='ctrl')
21         créer une action DupliquerNoeud
22       sinon
23         créer une action DeplacerNoeud
24    }}
25    retourner recycle
26 }}
```

Le feedback intérimaire de l'instrument *Main*, décrit par le pseudo-code suivant, change le curseur en fonction de l'action en cours.

```
1  feedbackInterimaire(){
2    si(action est DeplacerNoeud)
3      si(action.peutFaire())
4        mettre curseur "déplacement"
5      sinon
6        mettre curseur "impossible de déplacer"
7    sinon si(action est SupprimerNoeud)
8        mettre curseur "suppression"
9  }
```

Un instrument doit se mettre à jour à chaque changement d'état de l'interaction en cours (démarrage, évolution, fin et avortement). Le pseudo-code qui suit décrit les méthodes de l'instrument *Main* permettant ce processus.

```
1  interactionDémarre(Interaction i) {
2   si(instrument déjà actif) fin
3   si(i est PressionBouton et
4      cible de i est NoeudUI ou ArbreUI) {
5     créer une action SélectionnerNoeud
6     activer l'instrument
7   } sinon si(i est GlisserDeposer et
8       existe une action SelectionnerNoeud){
9     si(touche de i==null et source est NoeudUI
10       et cible==instrSupp.boutSupp)
11      créer une action DeplacerNoeud
12    activer l'instrument
13   }
14  }
15  interactionEvolue(Interaction i){
16   si(instrument pas actif ou i!=inter) fin
17   si(!recyclageActions(i) et action==null et
18    i est GlisserDeposer et touche de i=='ctrl'
19    et objet de i est NoeudUI)
20    créer une action DupliquerNoeud
21   majAction()
22  }
23  interactionTermine(Interaction i){
24   si(instrument pas actif ou i!=inter) fin
25   terminer l'action en cours
26   désactiver l'instrument
27   feedback intérimaire
28  }
29  interactionAvorte(Interaction i) {
30   si(instrument pas actif ou i!=inter) fin
31   avorter l'action en cours
32   désactiver l'instrument
33   feedback intérimaire
34 }}
```

IMPLEMENTATION

Notre implémentation de Malai est réalisée en Java et fait 5000 lignes de code (pour notre prototype actuel, commentaires exclus). En plus des classes nécessaires à la création d'instruments, d'interactions et d'actions spécifiques, notre prototype fournit un ensemble non-exhaustive d'interactions ainsi qu'un ensemble d'instruments et d'actions génériques. A l'aide de cette librairie, nous avons développé une application dédiée au dessin vectoriel (1300 lignes de code), ainsi que l'application présentée tout au long de cet article, permettant l'édition d'un fichier XML par le biais d'un arbre (1300 lignes de code) [1]. Au travers de l'éditeur de dessins, nous avons utilisé des interactions complexes telles que l'interaction bimanuelle pour redimensionner des formes, ainsi qu'une interaction multimodale composée d'un gyroscope (la Wiimote) contrôlé par la voix pour déplacer et pivoter des formes. Ces interactions ont été décrites et utilisées sans problème par des instruments *via* Malai. Le code de ces applications servira de patron pour la génération (semi-)automatique du code avec des outils IDM.

1. L'implémentation de Malai et les applications développées sont disponibles à l'adresse suivante : http ://gri.eseo.fr/software/malai/

EVALUATION

Dans cette section, nous comparons tout d'abord Malai à d'autres architectures et évaluons ensuite notre architecture en utilisant certains critères tirés de [20].

VIGO et M-CIU proposent respectivement, sur la base de l'interaction instrumentale, une architecture dédiée à la création d'IU distribuées [12] et un modèle proposant d'unifier les techniques d'interaction du type *glisser-déposer* et ses dérivés [7]. VIGO et M-CIU ne séparent pas les notions d'action et d'interaction de l'instrument. Un instrument est décrit par une unique interaction et peut être utilisé dans différents systèmes et IU. Dans Malai, c'est l'interaction qui est considérée comme indépendante et réutilisable et non l'instrument (des instruments génériques sont cependant fournis avec notre prototype pour être utilisés dans différents SI). VIGO et M-CIU n'utilisent pas d'action telle que définie dans Malai, perdant ainsi les avantages de notre modèle d'action, tels que le recyclage d'actions et la définition du feedback intérimaire d'une action. Cependant, ces inconvénients peuvent s'avérer avantageux en terme de simplicité d'utilisation comparés à notre modèle plus complexe à mettre en place mais qui permet la réutilisation d'interactions et d'actions. De plus, ils ne fournissent pas de langage dédié au lien entre les données et les présentations tel que notre langage Malan [5]. MPDC est une architecture qui étend MVC en sortant le processus de *picking*, récupérant des éléments d'une IU, du contrôleur, permettant à ce dernier d'être modulable pour différentes vues [8]. Les instruments et les interactions de notre modèle permettent cette modularité : les HID fournissent des paramètres aux instruments, lesquels les utilisent pour en déduire les éléments de l'IU concernés. De plus, MPDC n'utilise pas de langage dédié, comme notre langage Malan, pour lier les données à la présentation.

Nous évaluons maintenant Malai en utilisant quatre critères d'évaluation de SI définis par Olsen [20] : la *généralité* de la solution proposée, son *pouvoir d'expression*, sa *flexibilité* et la *proximité* entre les moyens utilisés pour exprimer des choix de conception et le problème à résoudre. Concernant la *généralité de la solution*, Malai permet la définition de SI de manière générale mais n'aborde pas les notions d'application distribuée, de réalité mixte ou d'interaction multimodale, ce qui, par conséquent, limite le *pouvoir d'expression* de Malan. Cependant, l'utilisation de Malai dans le cadre de l'interaction multimodale semble envisageable si l'on considère les gestes, la voix, *etc.* comme des HID fournissant des évènements à des interactions au même titre que le clavier ou la souris. Le pouvoir d'expression de Malai est également limité par rapport à d'autres formalismes tels que ICO qui permet de décrire de manière formelle certaines parties des SI à l'aide de réseaux de Pétri facilitant ainsi la détection d'erreurs de conception [17]. Grâce à l'interaction instrumentale, Malai permet l'utilisation d'interactions WIMP et post-WIMP [2]. La *flexibilité* de Malai consiste en la possibilité de réutiliser des actions pour différentes interfaces utilisant des mêmes données (le feedback intérimaire de chaque action doit tout de même être réécrit) et l'indépendance des interactions, lesquelles peuvent être utilisées dans différents SI. Dans notre modèle, nous utilisons les concepts d'action, d'interaction et d'instrument fondés sur la métaphore de l'outil du monde réel : un utilisateur interagit à l'aide d'outils (les instruments) sur des objets afin d'effectuer des actions. Il est ainsi possible de développer une partie d'un SI tout en raisonnant de manière naturelle ce qui peut favoriser la proximité entre le problème et les moyens utilisés.

VERS UNE APPROCHE FONDEE SUR L'INGENIERIE DES MODELES

Elever le niveau d'abstraction lors de la définition de systèmes n'est pas un problème nouveau. Le langage de modélisation UML [22] fournit un ensemble de diagrammes permettant la conception de manière abstraite de systèmes, dont le diagramme de classes utilisé, entre autres, pour générer automatiquement du code Java ou C#. L'ingénierie des modèles (IDM) [4] est un paradigme dont l'idée centrale est de considérer le cycle de développement d'un logiciel comme une chaîne de transformations de modèles de différents niveaux d'abstraction, où un modèle est une abstraction d'un système étudié. Des travaux en IHM reprennent ces principes en les appliquant aux problèmes de l'ingénierie [26, 21] des interfaces en se basant sur une architecture à quatre niveaux proposée par Cameleon [6] : le modèle de tâches, l'interface abstraite (indépendante de tout détail graphique), l'interface concrète (décrit les informations graphiques) et l'interface finale (le code du SI). Nos modèles Malai et Malan s'accordent avec cette architecture :

- l'interface abstraite regroupe le modèle de données du SI, sa présentation abstraite, le lien entre ces deux derniers défini avec Malan, et le modèle d'action sans le feedback intérimaire des actions, ni les actions dépendantes de la présentation concrète ;
- l'interface concrète complète l'interface abstraite en y ajoutant la présentation concrète, le lien entre les présentations abstraite et concrète, les instruments, les interactions, les feedbacks intérimaires des actions, et les actions dépendantes de la présentation concrète ;
- l'interface finale sera générée grâce à des transformations de modèles prédéfinies utilisant en entrée l'interface concrète et un choix de plate-forme d'application (Java, .NET, *etc.*).

CONCLUSION ET PERSPECTIVES

Cet article présente Malai, un modèle d'interaction conceptuel dédié au développement de Systèmes Interactifs (SI). Malai se compose de quatre modèles : les modèles d'interface, d'instrument, d'action et d'interaction. Il complète nos travaux précédents sur le modèle de données et son lien avec l'interface [5]. Malai vise à mettre au profit

les principes majeurs des interacteurs [15], de l'interaction instrumentale [2], de la manipulation directe [24], du modèle d'action de Norman [18] et du modèle DPI [3]. Malai permet une définition claire du feedback intérimaire des actions et des instruments, et fournit un nouveau modèle d'action visant à faciliter le développement de SI notamment grâce au recyclage d'actions. En rendant les interactions indépendantes des SI, une liste extensible d'interactions peut être ainsi utilisée pour la définition d'instrument dans différents SI.

Dans nos travaux futurs, des études seront réalisées pour comparer l'implémentation de Malai à d'autres boîtes à outils afin d'évaluer les bénéfices potentiels de notre modèle. Nous continuerons également l'utilisation de Malai et de Malan dans le contexte de l'ingénierie des modèles dans le but de générer des SI pour différentes plate-formes à partir de modèles abstraits et complets entre lesquels l'instrument jouera le rôle de pivot.

BIBLIOGRAPHIE

1. Appert, C., and Beaudouin-Lafon, M. SwingStates : adding state machines to Java and the Swing toolkit. *Softw., Pract. and Exper.*, 38(11) :1149–1182, 2008.
2. Beaudouin-Lafon, M. Instrumental interaction : An interaction model for designing post-WIMP interfaces. In *Proc. of CHI '00*, pages 446–453, 2000.
3. Beaudoux, O., and Beaudouin-Lafon, M. OpenDPI : A toolkit for developing document-centered environments. In *Enterprise Infor. Syst. VII*. Springer, 2006.
4. Bézivin, J. On the unification power of models. *Software and Systems Modeling*, 4(2) :171–188, 2005.
5. Blouin, A., Beaudoux, O., and Loiseau, S. Malan : A mapping language for the data manipulation. In *Proc. of DocEng'08*, pages 66–75. ACM, 2008.
6. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J. A unifying reference framework for multi-target user interfaces. *Interacting With Computers*, 15(3) :289–308, 2003.
7. Collomb, M., and Hascoët, M. Extending drag-and-drop to new interactive environments : A multi-display, multi-instrument and multi-user approach. *Interacting with Computers*, 20 :562–573, 2008.
8. Conversy, S., Barboni, E., Navarre, D., and Palanque, P. Improving modularity of interactive software with the MDPC architecture. In *Proc. of EIS '07*, 2007.
9. Crease, M., Gray, P., and Brewster, S. A toolkit of mechanism and context independent widgets. *Lecture Notes in Comp. Sc.*, 1946 :121–133, 2001.
10. Dragicevic, P., and Fekete, J. Input device selection and interaction configuration with ICON. In *Proc. of IHM-HCI 01*, pages 543–448, 2001.
11. Freeman, E., and Freeman, E. *Design Patterns*. O'Reilly, 2005.
12. Klokmose, C. N., and Beaudouin-Lafon, M. VIGO : instrumental interaction in multi-surface environments. In *Proc. of CHI '09*, pages 869–878, 2009.
13. Krasner, G. E., and Pope, S. T. A description of the model-view-controller user interface paradigm in smalltalk80 system. *J. of OOP*, 1 :26–49, 1988.
14. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., Florins, M., and Trevisan, D. UsiXML : a user interface description language for specifying multimodal user interfaces. In *Proc. of WMI'2004*, 2004.
15. Myers, B., Hudson, S. E., and Pausch, R. Past, present, and future of user interface software tools. *ACM Trans. on CHI*, 7(1) :3–28, 2000.
16. Myers, B. A. A new model for handling input. *ACM Trans. on Information Systems*, 8(3) :289–320, 1990.
17. Navarre, D., Palanque, P., Bastide, R., and Sy, O. Structuring interactive systems specifications for executability and prototypability. *Lecture Notes in Computer Science*, 1946 :97–119, 2001.
18. Norman, D. A., and Draper, S. W. *User-Centered System Design : New Perspectives on Human-Computer Interaction*. Lawrence Erlbaum Associates, 1986.
19. OASIS. User interface markup language (UIML). Technical report, OASIS, 2008.
20. Olsen, Jr., D. R. Evaluating user interface systems research. In *Proc. of UIST '07*, pages 251–258, 2007.
21. Paternò, F., Santoro, C., Mäntyjärvi, J., Mori, G., and Sansone, S. Authoring pervasive multimodal user interfaces. *Int. J. Web Engineering and Technology*, 4(2) :235–261, 2008.
22. Rumbaugh, J., Jacobson, I., and Booch, G. *The Unified Modeling Language Reference Manual*. Addison Wesley Professional, 2004.
23. Sells, C., and Griffiths, I. *Programming Windows Presentation Foundation*. O'Reilly, 2005.
24. Shneiderman, B. Direct manipulation : a step beyond programming languages. *IEEE Computer*, 16(8) :57–69, 1983.
25. Shneiderman, B. *Designing the User Interface : Strategies for Effective Human-Computer Interaction*. Addison-Wesley, 2004.
26. Silva, P. P. D. User interface declarative models and development environments : A survey. In *Proc. of DSV-IS2000*, pages 207–226. Springer-Verlag, 2000.
27. Vanderdonckt, J. A MDA-compliant environment for developing user interfaces of information systems. *Lecture Notes in Computer Science*, 3520/2005 :16–31, 2005.